sirshkonar01@gmail.com_Facebook_Case_Study_v1

April 25, 2019

Social network Graph Link Prediction - Facebook Challenge

0.0.1 Problem statement:

Given a directed social graph, have to predict missing links to recommend users (Link Prediction in graph)

0.0.2 Data Overview

Taken data from facebook's recruting challenge on kaggle https://www.kaggle.com/c/FacebookRecruiting
data contains two columns source and destination eac edge in graph - Data columns (total 2 columns):
- source_node int64
- destination_node int64

0.0.3 Mapping the problem into supervised learning problem:

- Generated training samples of good and bad links from given directed graph and for each link got some features like no of followers, is he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml model based on these features to predict link.
- Some reference papers and videos :
  - https://www.cs.cornell.edu/home/kleinber/link-pred.pdf
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf
  - https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf
  - https://www.youtube.com/watch?v=2M77Hgy17cg

0.0.4 Business objectives and constraints:

- No low-latency requirement.
- Probability of prediction is useful to recommend ighest probability links

0.0.5 Performance metric for supervised learning:

- Both precision and recall is important so F1 score is good choice

- Confusion matrix

```
In [2]: #Importing Libraries
        # please do go through this python notebook:
        import warnings
        warnings.filterwarnings("ignore")

        import csv
        import pandas as pd#pandas to create small dataframes
        import datetime #Convert to unix time
        import time #Convert to unix time
        # if numpy is not installed already : pip3 install numpy
        import numpy as np#Do aritmetic operations on arrays
        # matplotlib: used to plot graphs
        import matplotlib
        import matplotlib.pylab as plt
        import seaborn as sns#Plots
        from matplotlib import rcParams#Size of plots
        from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
        import math
        import pickle
        import os
        # to install xgboost: pip3 install xgboost
        import xgboost as xgb

        import warnings
        import networkx as nx
        import pdb
        import pickle
```

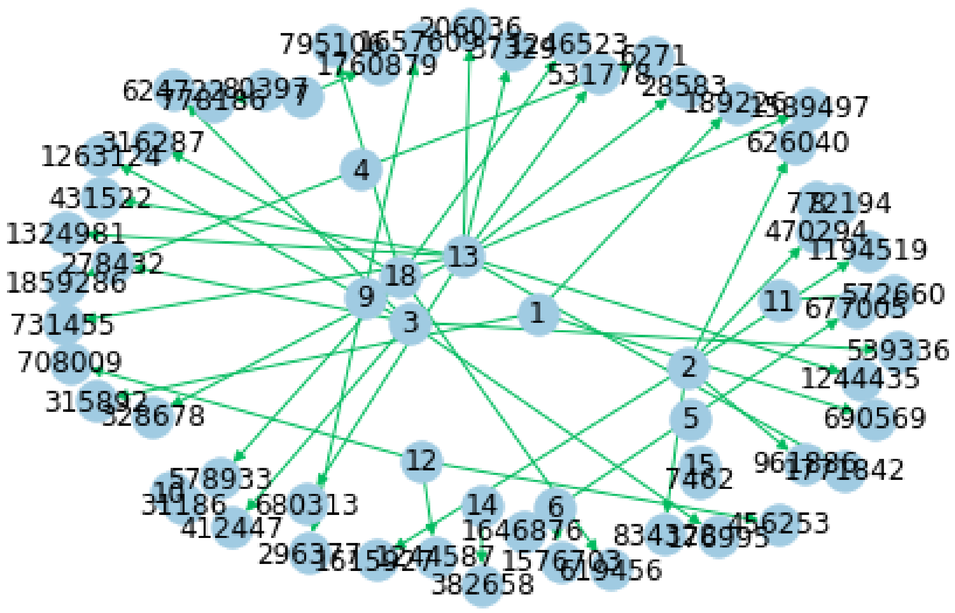
```
In [2]: #reading graph
        if not os.path.isfile('data/after_eda/train_woheader.csv'):
            traincsv = pd.read_csv('data/train.csv')
            print(traincsv[traincsv.isna().any(1)])
            print(traincsv.info())
            print("Number of diplicate entries: ",sum(traincsv.duplicated()))
            traincsv.to_csv('data/after_eda/train_woheader.csv',header=False,index=False)
            print("saved the graph into file")
        else:
            g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=',',create_using=n
            print(nx.info(g))
```

```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
```

Displaying a sub graph

```
In [3]: if not os.path.isfile('train_woheader_sample.csv'):
            pd.read_csv('data/train.csv', nrows=50).to_csv('train_woheader_sample.csv',header=F

        subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',create_using=nx.Di(
        # https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with-networkx-and-ma

        pos=nx.spring_layout(subgraph)
        nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm
        plt.savefig("graph_sample.pdf")
        print(nx.info(subgraph))
```

```
Name:
Type: DiGraph
Number of nodes: 66
Number of edges: 50
Average in degree:   0.7576
Average out degree:   0.7576
```



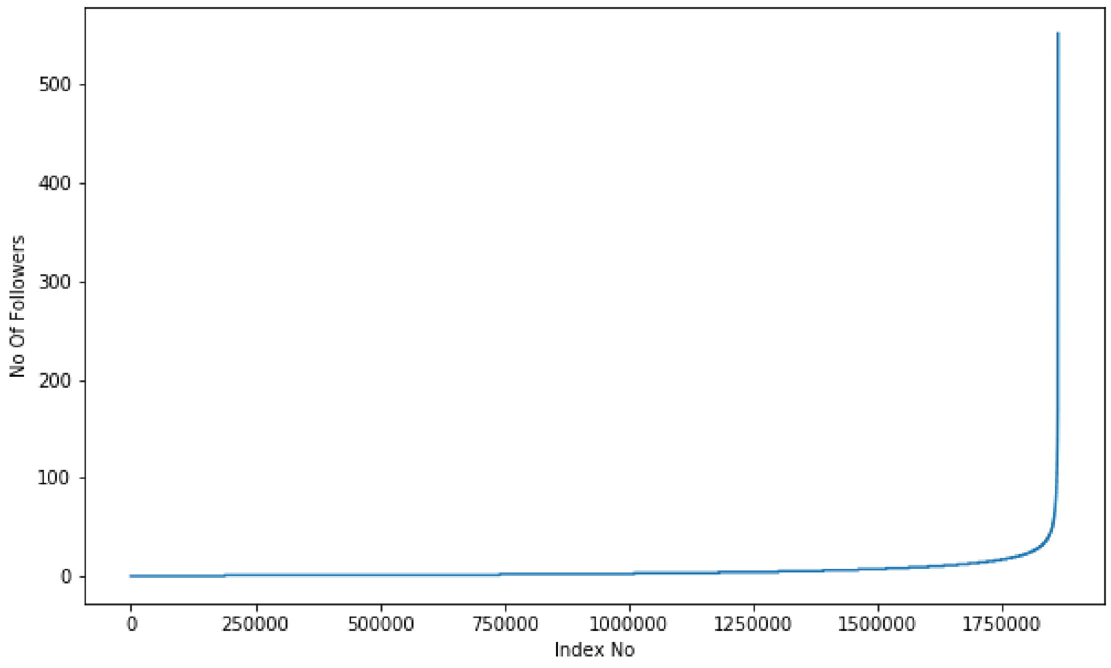
1 1. Exploratory Data Analysis

```
In [4]: # No of Unique persons
        print("The number of unique persons",len(g.nodes()))
```

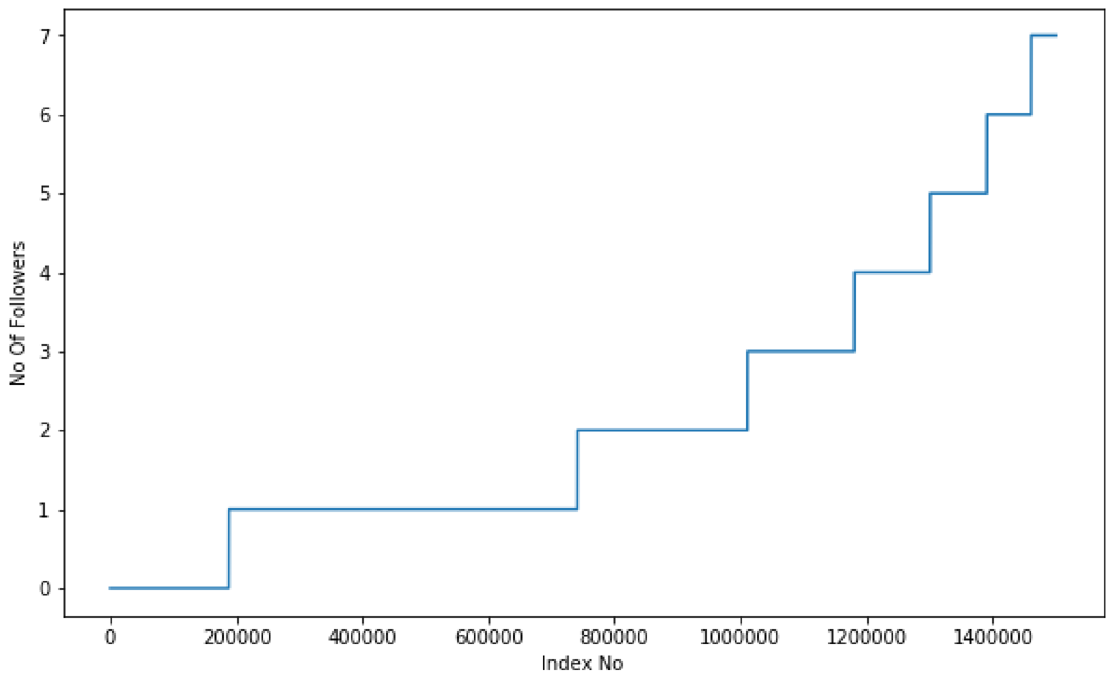
```
The number of unique persons 1862220
```

1.1 1.1 No of followers for each person

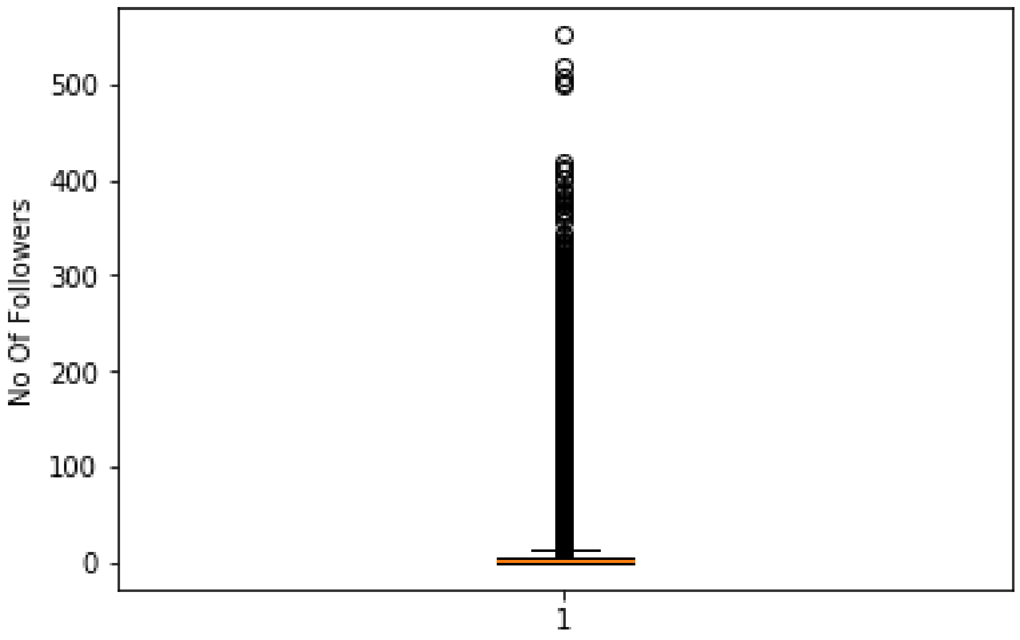
```
In [5]: indegree_dist = list(dict(g.in_degree()).values())
        indegree_dist.sort()
        plt.figure(figsize=(10,6))
        plt.plot(indegree_dist)
        plt.xlabel('Index No')
        plt.ylabel('No Of Followers')
        plt.show()
```



```
In [6]: indegree_dist = list(dict(g.in_degree()).values())
        indegree_dist.sort()
        plt.figure(figsize=(10,6))
        plt.plot(indegree_dist[0:1500000])
        plt.xlabel('Index No')
        plt.ylabel('No Of Followers')
        plt.show()
```



```
In [7]: plt.boxplot(indegree_dist)
        plt.ylabel('No Of Followers')
        plt.show()
```



```
In [8]: ### 90-100 percentile
        for i in range(0,11):
            print(90+i,'percentile value is',np.percentile(indegree_dist,90+i))
```

```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 552.0
```

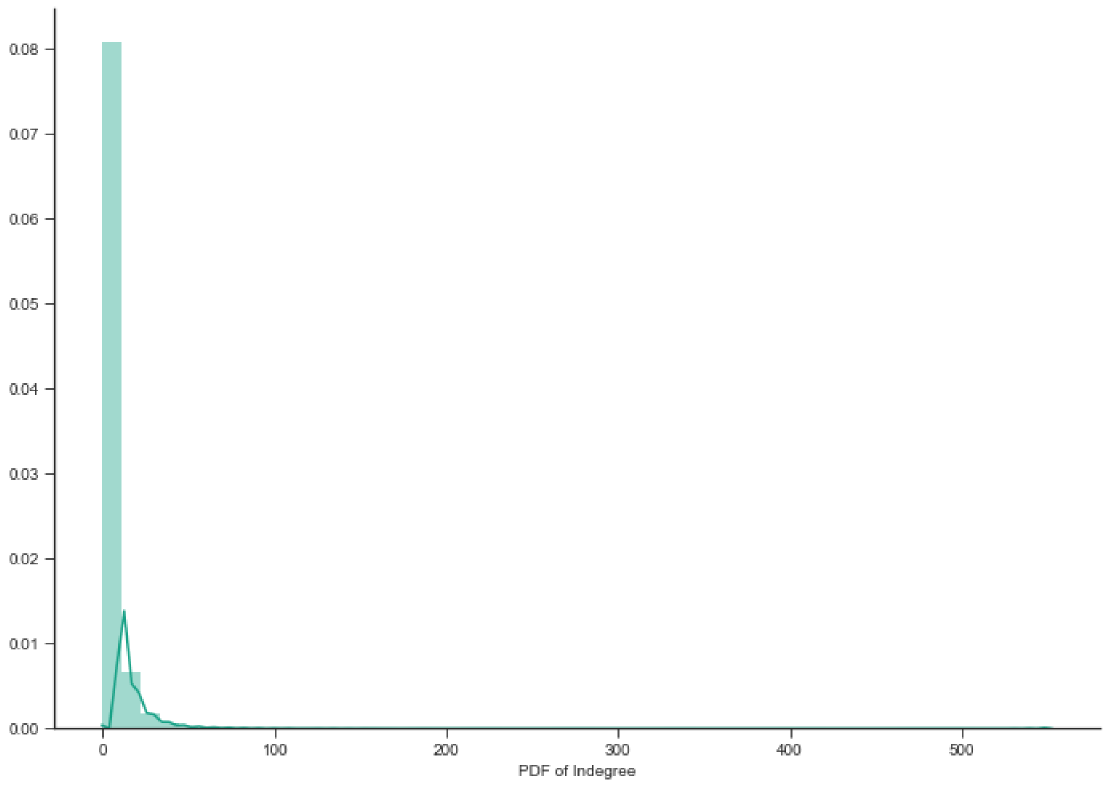
99% of data having followers of 40 only.

```
In [9]: ### 99-100 percentile
        for i in range(10,110,10):
            print(99+(i/100),'percentile value is',np.percentile(indegree_dist,99+(i/100)))
```

```
99.1 percentile value is 42.0
99.2 percentile value is 44.0
99.3 percentile value is 47.0
99.4 percentile value is 50.0
99.5 percentile value is 55.0
99.6 percentile value is 61.0
99.7 percentile value is 70.0
99.8 percentile value is 84.0
99.9 percentile value is 112.0
100.0 percentile value is 552.0
```

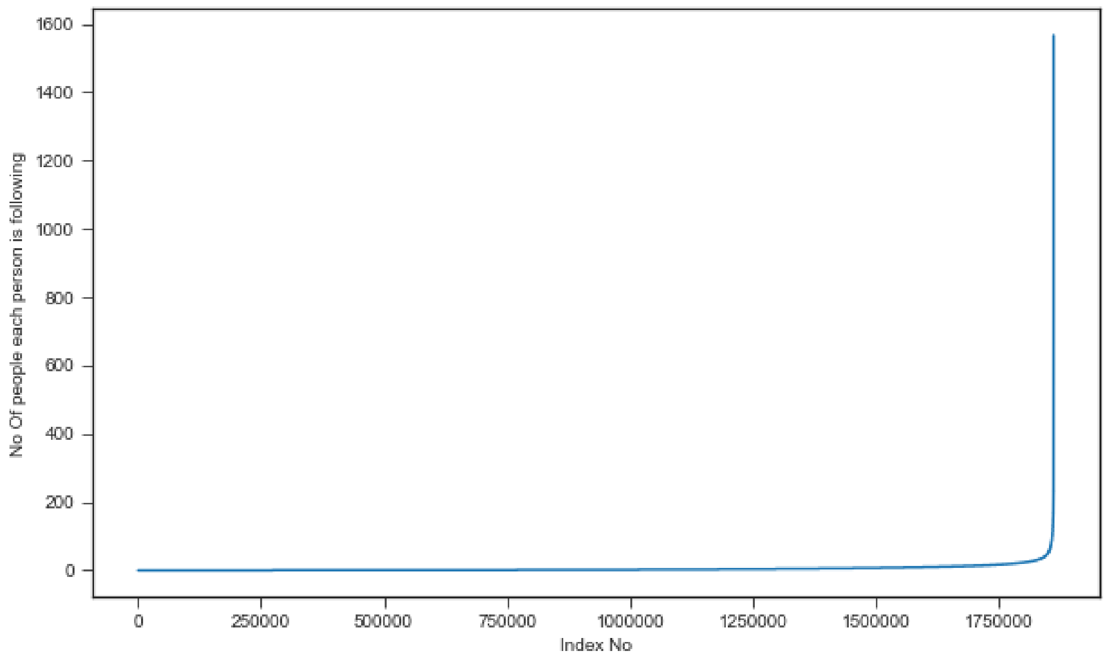
```
In [10]: %matplotlib inline
         sns.set_style('ticks')
         fig, ax = plt.subplots()
         fig.set_size_inches(11.7, 8.27)
         sns.distplot(indegree_dist, color='#16A085')
         plt.xlabel('PDF of Indegree')
         sns.despine()
         #plt.show()
```

```
D:\installed\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6571: UserWarning: The 'norme
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

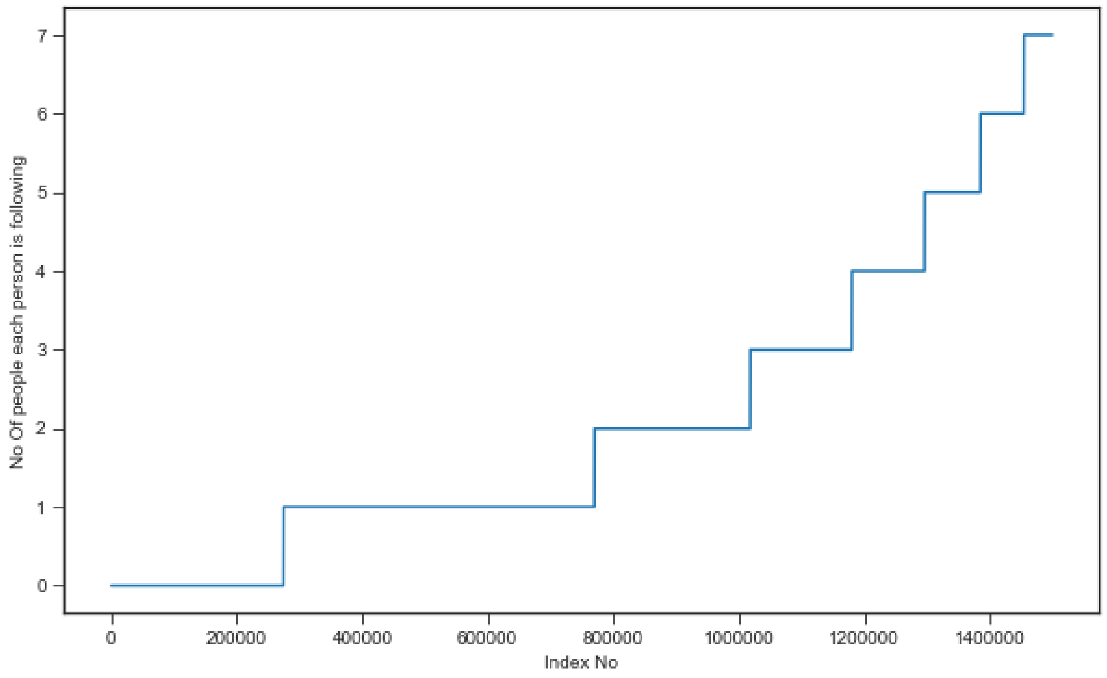


1.2 1.2 No of people each person is following

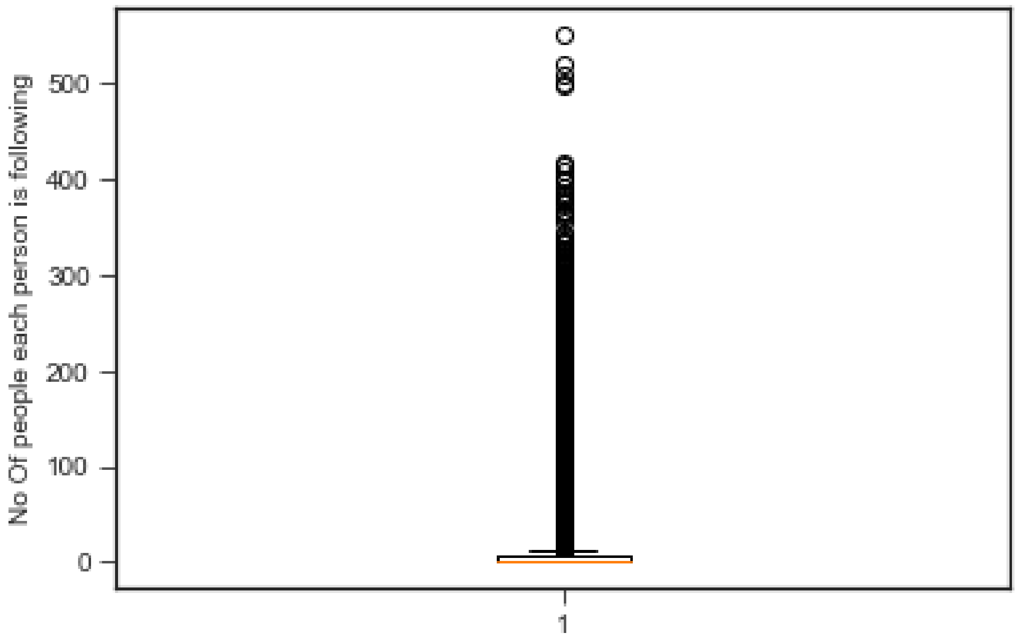
```
In [11]: outdegree_dist = list(dict(g.out_degree()).values())
         outdegree_dist.sort()
         plt.figure(figsize=(10,6))
         plt.plot(outdegree_dist)
         plt.xlabel('Index No')
         plt.ylabel('No Of people each person is following')
         plt.show()
```



```
In [12]: indegree_dist = list(dict(g.in_degree()).values())
         indegree_dist.sort()
         plt.figure(figsize=(10,6))
         plt.plot(outdegree_dist[0:1500000])
         plt.xlabel('Index No')
         plt.ylabel('No Of people each person is following')
         plt.show()
```



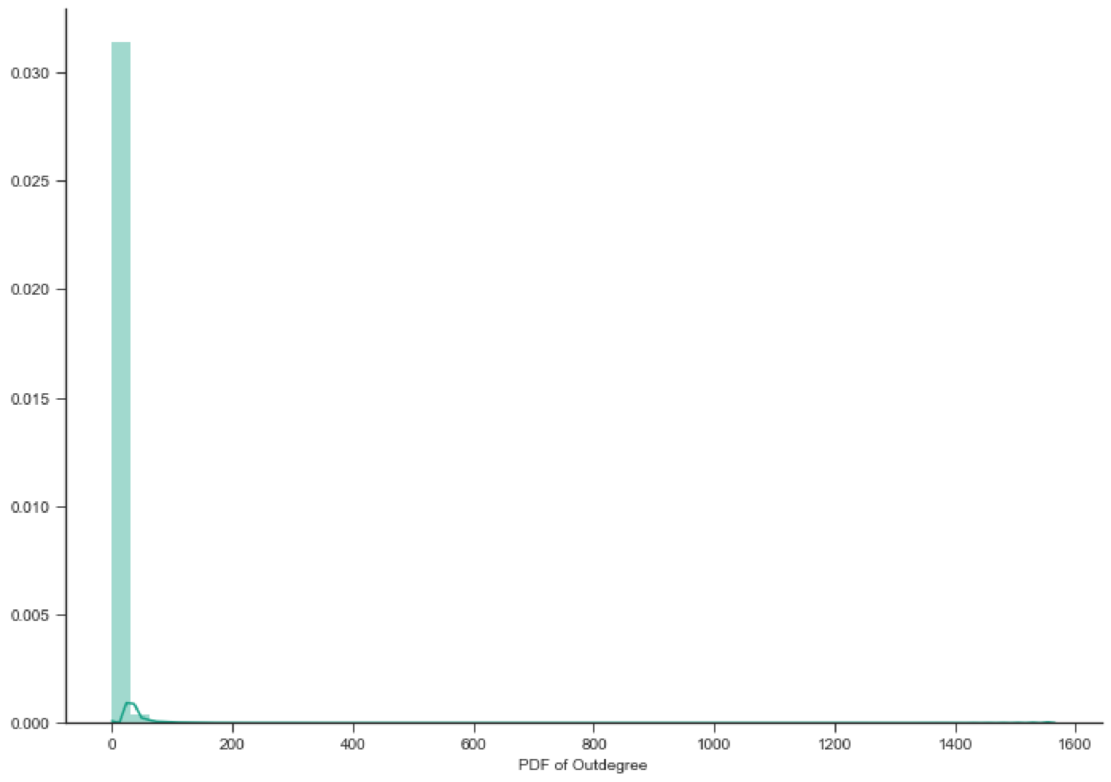
```
In [13]: plt.boxplot(indegree_dist)
         plt.ylabel('No Of people each person is following')
         plt.show()
```



```
In [14]: ### 90-100 percentile
         for i in range(0,11):
             print(90+i,'percentile value is',np.percentile(outdegree_dist,90+i))

90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 1566.0
```

```
In [15]: ### 99-100 percentile
         for i in range(10,110,10):
             print(99+(i/100),'percentile value is',np.percentile(outdegree_dist,99+(i/100)))

99.1 percentile value is 42.0
99.2 percentile value is 45.0
99.3 percentile value is 48.0
99.4 percentile value is 52.0
99.5 percentile value is 56.0
99.6 percentile value is 63.0
99.7 percentile value is 73.0
99.8 percentile value is 90.0
99.9 percentile value is 123.0
100.0 percentile value is 1566.0
```

```
In [16]: sns.set_style('ticks')
         fig, ax = plt.subplots()
         fig.set_size_inches(11.7, 8.27)
         sns.distplot(outdegree_dist, color='#16A085')
         plt.xlabel('PDF of Outdegree')
         sns.despine()

D:\installed\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6571: UserWarning: The 'norm
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



```
In [17]: print('No of persons those are not following anyone are' ,sum(np.array(outdegree_dist
                                                sum(np.array(outdegree_dist)==0)*100/len(outdegree_dis
```

```
No of persons those are not following anyone are 274512 and % is 14.741115442858524
```

```
In [18]: print('No of persons having zero followers are' ,sum(np.array(indegree_dist)==0),'and
                                                sum(np.array(indegree_dist)==0)*100/len(indegree_dist
```

```
No of persons having zero followers are 188043 and % is 10.097786512871734
```

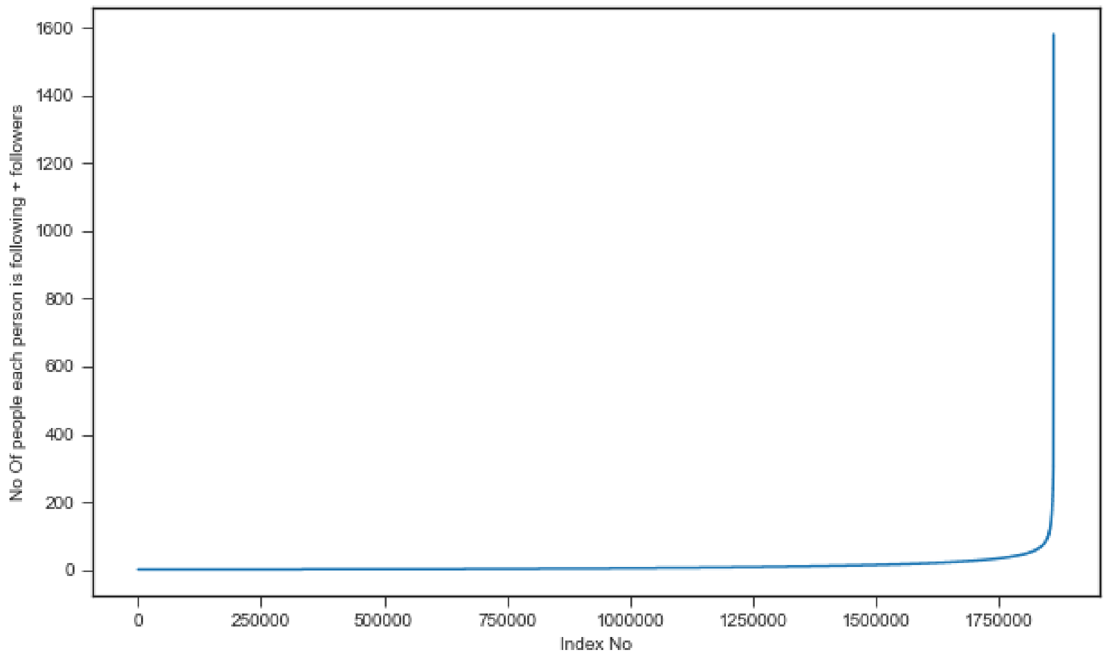
```
In [19]: count=0
         for i in g.nodes():
             if len(list(g.predecessors(i)))==0 :
                 if len(list(g.successors(i)))==0:
                     count+=1
         print('No of persons those are not not following anyone and also not having any follo
```

```
No of persons those are not not following anyone and also not having any followers are 0
```

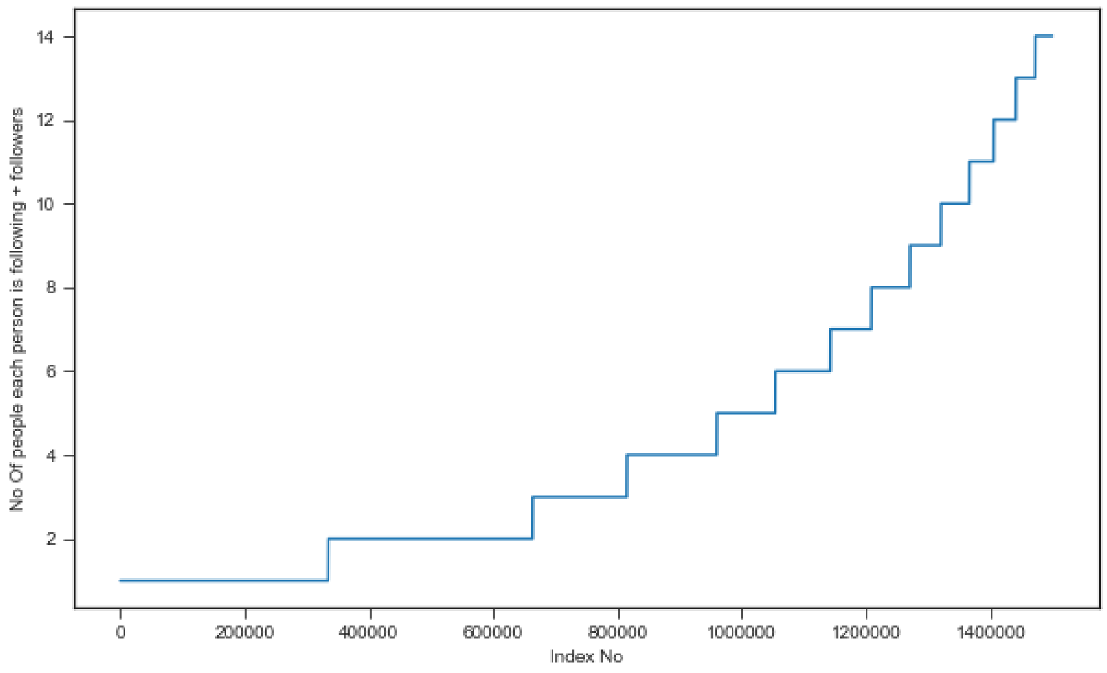
1.3 1.3 both followers + following

```
In [20]: from collections import Counter
         dict_in = dict(g.in_degree())
         dict_out = dict(g.out_degree())
         d = Counter(dict_in) + Counter(dict_out)
         in_out_degree = np.array(list(d.values()))
```

```
In [21]: in_out_degree_sort = sorted(in_out_degree)
         plt.figure(figsize=(10,6))
         plt.plot(in_out_degree_sort)
         plt.xlabel('Index No')
         plt.ylabel('No Of people each person is following + followers')
         plt.show()
```



```
In [22]: in_out_degree_sort = sorted(in_out_degree)
         plt.figure(figsize=(10,6))
         plt.plot(in_out_degree_sort[0:1500000])
         plt.xlabel('Index No')
         plt.ylabel('No Of people each person is following + followers')
         plt.show()
```



```
In [23]: ### 90-100 percentile
         for i in range(0,11):
             print(90+i,'percentile value is',np.percentile(in_out_degree_sort,90+i))
```

```
90 percentile value is 24.0
91 percentile value is 26.0
92 percentile value is 28.0
93 percentile value is 31.0
94 percentile value is 33.0
95 percentile value is 37.0
96 percentile value is 41.0
97 percentile value is 48.0
98 percentile value is 58.0
99 percentile value is 79.0
100 percentile value is 1579.0
```

```
In [24]: ### 99-100 percentile
         for i in range(10,110,10):
             print(99+(i/100),'percentile value is',np.percentile(in_out_degree_sort,99+(i/100)
```

```
99.1 percentile value is 83.0
99.2 percentile value is 87.0
99.3 percentile value is 93.0
99.4 percentile value is 99.0
```

```
99.5 percentile value is 108.0
99.6 percentile value is 120.0
99.7 percentile value is 138.0
99.8 percentile value is 168.0
99.9 percentile value is 221.0
100.0 percentile value is 1579.0
```

```
In [25]: print('Min of no of followers + following is',in_out_degree.min())
         print(np.sum(in_out_degree==in_out_degree.min()),' persons having minimum no of follow
```

```
Min of no of followers + following is 1
334291  persons having minimum no of followers + following
```

```
In [26]: print('Max of no of followers + following is',in_out_degree.max())
         print(np.sum(in_out_degree==in_out_degree.max()),' persons having maximum no of follow
```

```
Max of no of followers + following is 1579
1  persons having maximum no of followers + following
```

```
In [27]: print('No of persons having followers + following less than 10 are',np.sum(in_out_degr
```

```
No of persons having followers + following less than 10 are 1320326
```

```
In [28]: print('No of weakly connected components',len(list(nx.weakly_connected_components(g)))
         count=0
         for i in list(nx.weakly_connected_components(g)):
             if len(i)==2:
                 count+=1
         print('weakly connected components wit 2 nodes',count)
```

```
No of weakly connected components 45558
weakly connected components wit 2 nodes 32195
```

2 2. Posing a problem as classification problem

2.1 2.1 Generating some edges which are not present in graph for supervised learning

Generated Bad links from graph which are not in graph and whose shortest path is greater than 2.

```
In [46]: %%time
         ###generating bad edges from given graph
         import random
         if not os.path.isfile('data/after_eda/missing_edges_final.p'):
             #getting all set of edges
```

```
        r = csv.reader(open('data/after_eda/train_woheader.csv','r'))
        edges = dict()
        for edge in r:
            edges[(edge[0], edge[1])] = 1


        missing_edges = set([])
        while (len(missing_edges)<9437519):
            a=random.randint(1, 1862220)
            b=random.randint(1, 1862220)
            tmp = edges.get((a,b),-1)
            if tmp == -1 and a!=b:
                try:
                    if nx.shortest_path_length(g,source=a,target=b) > 2:

                        missing_edges.add((a,b))
                    else:
                        continue
                except:
                        missing_edges.add((a,b))
            else:
                continue
        pickle.dump(missing_edges,open('data/after_eda/missing_edges_final.p','wb'))
    else:
        missing_edges = pickle.load(open('data/after_eda/missing_edges_final.p','rb'))
```

```
Wall time: 5.08 s
```

```
In [47]: len(missing_edges)
```

```
Out[47]: 9437519
```

2.2 2.2 Training and Test data split:

Removed edges from Graph and used as test data and after removing used that graph for creating features for Train and test data

```
In [48]: from sklearn.model_selection import train_test_split
         if (not os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (not os.path.is
             #reading total data df
             df_pos = pd.read_csv('data/train.csv')
             df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_n

             print("Number of nodes in the graph with edges", df_pos.shape[0])
             print("Number of nodes in the graph without edges", df_neg.shape[0])

             #Trian test split
             #Spiltted data into 80-20
```

```
        #positive links and negative links seperatly because we need positive training da
        #and for feature generation
        X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.one
        X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zer

        print('='*60)
        print("Number of nodes in the train data graph with edges", X_train_pos.shape[0],"
        print("Number of nodes in the train data graph without edges", X_train_neg.shape[0
        print('='*60)
        print("Number of nodes in the test data graph with edges", X_test_pos.shape[0],"="
        print("Number of nodes in the test data graph without edges", X_test_neg.shape[0]

        #removing header and saving
        X_train_pos.to_csv('data/after_eda/train_pos_after_eda.csv',header=False, index=Fa
        X_test_pos.to_csv('data/after_eda/test_pos_after_eda.csv',header=False, index=Fals
        X_train_neg.to_csv('data/after_eda/train_neg_after_eda.csv',header=False, index=Fa
        X_test_neg.to_csv('data/after_eda/test_neg_after_eda.csv',header=False, index=Fals
    else:
        #Graph from Traing data only
        del missing_edges
```

```
Number of nodes in the graph with edges 9437519
Number of nodes in the graph without edges 9437519
============================================================
Number of nodes in the train data graph with edges 7550015 = 7550015
Number of nodes in the train data graph without edges 7550015 = 7550015
============================================================
Number of nodes in the test data graph with edges 1887504 = 1887504
Number of nodes in the test data graph without edges 1887504 = 1887504
```

```
In [49]: if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.path.isfile('da
             train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter='
             test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv',delimiter=','
             print(nx.info(train_graph))
             print(nx.info(test_graph))

             # finding the unique nodes in the both train and test graphs
             train_nodes_pos = set(train_graph.nodes())
             test_nodes_pos = set(test_graph.nodes())

             trY_teY = len(train_nodes_pos.intersection(test_nodes_pos))
             trY_teN = len(train_nodes_pos - test_nodes_pos)
             teY_trN = len(test_nodes_pos - train_nodes_pos)

             print('no of people common in train and test -- ',trY_teY)
             print('no of people present in train but not present in test -- ',trY_teN)
```

```
        print('no of people present in test but not present in train -- ',teY_trN)
        print(' % of people not there in Train but exist in Test in total Test data are {
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
Name:
Type: DiGraph
Number of nodes: 1144623
Number of edges: 1887504
Average in degree:   1.6490
Average out degree:   1.6490
no of people common in train and test --  1063125
no of people present in train but not present in test --  717597
no of people present in test but not present in train --  81498
 % of people not there in Train but exist in Test in total Test data are 7.1200735962845405 %
```

we have a cold start problem here

```
In [50]: #final train and test data sets
         if (not os.path.isfile('data/after_eda/train_after_eda.csv')) and \
         (not os.path.isfile('data/after_eda/test_after_eda.csv')) and \
         (not os.path.isfile('data/train_y.csv')) and \
         (not os.path.isfile('data/test_y.csv')) and \
         (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and \
         (os.path.isfile('data/after_eda/test_pos_after_eda.csv')) and \
         (os.path.isfile('data/after_eda/train_neg_after_eda.csv')) and \
         (os.path.isfile('data/after_eda/test_neg_after_eda.csv')):

             X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv', names=['sourc
             X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_
             X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['sourc
             X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', names=['source_

             print('='*60)
             print("Number of nodes in the train data graph with edges", X_train_pos.shape[0])
             print("Number of nodes in the train data graph without edges", X_train_neg.shape[
             print('='*60)
             print("Number of nodes in the test data graph with edges", X_test_pos.shape[0])
             print("Number of nodes in the test data graph without edges", X_test_neg.shape[0]

             X_train = X_train_pos.append(X_train_neg,ignore_index=True)
             y_train = np.concatenate((y_train_pos,y_train_neg))
             X_test = X_test_pos.append(X_test_neg,ignore_index=True)
```

```
            y_test = np.concatenate((y_test_pos,y_test_neg))

            X_train.to_csv('data/after_eda/train_after_eda.csv',header=False,index=False)
            X_test.to_csv('data/after_eda/test_after_eda.csv',header=False,index=False)
            pd.DataFrame(y_train.astype(int)).to_csv('data/train_y.csv',header=False,index=Fa
            pd.DataFrame(y_test.astype(int)).to_csv('data/test_y.csv',header=False,index=Fals
```

```
============================================================
Number of nodes in the train data graph with edges 7550015
Number of nodes in the train data graph without edges 7550015
============================================================
Number of nodes in the test data graph with edges 1887504
Number of nodes in the test data graph without edges 1887504
```

```
In [51]: print("Data points in train data",X_train.shape)
         print("Data points in test data",X_test.shape)
         print("Shape of traget variable in train",y_train.shape)
         print("Shape of traget variable in test", y_test.shape)
```

```
Data points in train data (15100030, 2)
Data points in test data (3775008, 2)
Shape of traget variable in train (15100030,)
Shape of traget variable in test (3775008,)
```

3 2. Featurizations

```
In [1]: import warnings
        warnings.filterwarnings("ignore")

        import csv
        import pandas as pd#pandas to create small dataframes
        import datetime #Convert to unix time
        import time #Convert to unix time
        # if numpy is not installed already : pip3 install numpy
        import numpy as np#Do aritmetic operations on arrays
        # matplotlib: used to plot graphs
        import matplotlib
        import matplotlib.pylab as plt
        import seaborn as sns#Plots
        from matplotlib import rcParams#Size of plots
        from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
        import math
        import pickle
        import os
        # to install xgboost: pip3 install xgboost
        import xgboost as xgb
        import warnings
        import networkx as nx
        import pdb
        import pickle
        from pandas import HDFStore,DataFrame
        from pandas import read_hdf
        from scipy.sparse.linalg import svds, eigs
        import gc
        from tqdm import tqdm
```

4 Reading Data

```
In [2]: if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
            train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',
            print(nx.info(train_graph))
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

4.1 2.1 Similarity measures

4.1.1 2.1.1 Jaccard Distance:

http://www.statisticshowto.com/jaccard-index/

$$j = \frac{|X \cap Y|}{|X \cup Y|} \tag{1}$$

```
In [3]: #for followees
        def jaccard_for_followees(a,b):
            try:
                if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(
                    return 0
                sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.success
                                            (len(set(train_graph.successors(a)).union(set(trai
            except:
                return 0
            return sim
```

```
In [4]: #one test case
        print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

```
In [5]: #node 1635354 not in graph
        print(jaccard_for_followees(273084,1505602))

0.0
```

```
In [6]: #for followers
        def jaccard_for_followers(a,b):
            try:
                if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecessors(b))) ==
                    return 0
                sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.prede
                                       (len(set(train_graph.predecessors(a)).union(set(train_
                return sim
            except:
                return 0
```

```
In [7]: print(jaccard_for_followers(273084,470294))

0
```

```
In [8]: #node 1635354 not in graph
        print(jaccard_for_followees(669354,1635354))

0
```

4.1.2 2.2 Cosine distance

$$CosineDistance = \frac{|X \cap Y|}{|X| \cdot |Y|} \tag{2}$$

```
In [9]: #for followees
        def cosine_for_followees(a,b):
            try:
                if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(
                    return 0
                sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.success
                                          (math.sqrt(len(set(train_graph.successors(a)))*len
                return sim
            except:
                return 0
```

```
In [10]: print(cosine_for_followees(273084,1505602))

0.0
```

```
In [11]: print(cosine_for_followees(273084,1635354))
```

```
0
```

```
In [12]: def cosine_for_followers(a,b):
             try:

                 if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecess
                     return 0
                 sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.prede
                                              (math.sqrt(len(set(train_graph.predecessors(a)))
                 return sim
             except:
                 return 0
```

```
In [13]: print(cosine_for_followers(2,470294))
```

```
0.02886751345948129
```

```
In [14]: print(cosine_for_followers(669354,1635354))
```

```
0
```

4.2 2.2 Ranking Measures

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_an

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links.

Mathematical PageRanks for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. (The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.

4.2.1 2.2.1 Page Ranking

https://en.wikipedia.org/wiki/PageRank

```
In [15]: if not os.path.isfile('data/fea_sample/page_rank.p'):
             pr = nx.pagerank(train_graph, alpha=0.85)
             pickle.dump(pr,open('data/fea_sample/page_rank.p','wb'))
         else:
             pr = pickle.load(open('data/fea_sample/page_rank.p','rb'))
```

```
In [16]: print('min',pr[min(pr, key=pr.get)])
         print('max',pr[max(pr, key=pr.get)])
         print('mean',float(sum(pr.values())) / len(pr))
```

```
min 1.6556497245737814e-07
max 2.7098251341935827e-05
mean 5.615699699389075e-07
```

```
In [17]: #for imputing to nodes which are not there in Train data
         mean_pr = float(sum(pr.values())) / len(pr)
         print(mean_pr)
```

```
5.615699699389075e-07
```

4.3 2.3 Other Graph Features

4.3.1 2.3.1 Shortest path:

```
In [18]: #if has direct edge then deleting that edge and calculating shortest path
         def compute_shortest_path_length(a,b):
             p=-1
             try:
                 if train_graph.has_edge(a,b):
                     train_graph.remove_edge(a,b)
                     p= nx.shortest_path_length(train_graph,source=a,target=b)
                     train_graph.add_edge(a,b)
                 else:
                     p= nx.shortest_path_length(train_graph,source=a,target=b)
                 return p
             except:
                 return -1
```

```
In [19]: #testing
         compute_shortest_path_length(77697, 826021)
```

```
Out[19]: 10
```

```
In [20]: #testing
         compute_shortest_path_length(669354,1635354)
```

```
Out[20]: -1
```

4.3.2 2.3.2 Checking for same community

```
In [21]: #getting weekly connected edges from graph
         wcc=list(nx.weakly_connected_components(train_graph))
         def belongs_to_same_wcc(a,b):
             index = []
```

```
            if train_graph.has_edge(b,a):
                return 1
            if train_graph.has_edge(a,b):
                    for i in wcc:
                        if a in i:
                            index= i
                            break
                    if (b in index):
                        train_graph.remove_edge(a,b)
                        if compute_shortest_path_length(a,b)==-1:
                            train_graph.add_edge(a,b)
                            return 0
                        else:
                            train_graph.add_edge(a,b)
                            return 1
                    else:
                        return 0
            else:
                    for i in wcc:
                        if a in i:
                            index= i
                            break
                    if(b in index):
                        return 1
                    else:
                        return 0
```

```
In [22]: belongs_to_same_wcc(861, 1659750)
```

```
Out[22]: 0
```

```
In [23]: belongs_to_same_wcc(669354,1635354)
```

```
Out[23]: 0
```

4.3.3 2.3.3 Adamic/Adar Index:

Adamic/Adar measures is defined as inverted sum of degrees of common neighbours for given two vertices.

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log(|N(u)|)}$$

```
In [24]: #adar index
        def calc_adar_in(a,b):
            sum=0
            try:
                n=list(set(train_graph.successors(a)).intersection(set(train_graph.successors
                if len(n)!=0:
                    for i in n:
```

```
                    sum=sum+(1/np.log10(len(list(train_graph.predecessors(i)))))
                return sum
            else:
                return 0
        except:
            return 0
```

```
In [25]: calc_adar_in(1,189226)
```

```
Out[25]: 0
```

```
In [26]: calc_adar_in(669354,1635354)
```

```
Out[26]: 0
```

4.3.4 2.3.4 Is the person following back?

```
In [27]: def follows_back(a,b):
             if train_graph.has_edge(b,a):
                 return 1
             else:
                 return 0
```

```
In [28]: follows_back(1,189226)
```

```
Out[28]: 1
```

```
In [29]: follows_back(669354,1635354)
```

```
Out[29]: 0
```

4.3.5 2.3.5 Katz Centrality:

https://en.wikipedia.org/wiki/Katz_centrality
https://www.geeksforgeeks.org/katz-centrality-centrality-measure/ Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node `i` is

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where `A` is the adjacency matrix of the graph G with eigenvalues

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

$$\alpha < \frac{1}{\lambda_{max}}.$$

```
In [30]: if not os.path.isfile('data/fea_sample/katz.p'):
             katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
             pickle.dump(katz,open('data/fea_sample/katz.p','wb'))
         else:
             katz = pickle.load(open('data/fea_sample/katz.p','rb'))
```

```
In [31]: print('min',katz[min(katz, key=katz.get)])
         print('max',katz[max(katz, key=katz.get)])
         print('mean',float(sum(katz.values())) / len(katz))
```

```
min 0.0007313532484065916
max 0.003394554981699122
mean 0.0007483800935562018
```

```
In [32]: mean_katz = float(sum(katz.values())) / len(katz)
         print(mean_katz)
```

```
0.0007483800935562018
```

4.3.6 2.3.6 Hits Score

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.
https://en.wikipedia.org/wiki/HITS_algorithm

```
In [33]: if not os.path.isfile('data/fea_sample/hits.p'):
             hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True
             pickle.dump(hits,open('data/fea_sample/hits.p','wb'))
         else:
             hits = pickle.load(open('data/fea_sample/hits.p','rb'))
```

```
In [34]: print('min',hits[0][min(hits[0], key=hits[0].get)])
         print('max',hits[0][max(hits[0], key=hits[0].get)])
         print('mean',float(sum(hits[0].values())) / len(hits[0]))
```

```
min 0.0
max 0.004868653378780953
mean 5.615699699344123e-07
```

4.3.7 2.3.7 Preferential Attachment

One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how ”rich” our two vertices are by calculating the multiplication between the number of friends (|(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity. http://be.amazd.com/link-prediction/

```
In [3]: def compute_pref_at(a, b):
            try:
                pa = len(set(train_graph.neighbors(a)))*len(set(train_graph.neighbors(b)))
            except:
                return 0
            return pa
```

```
In [4]: #Testing function
        compute_pref_at(1, 189226)
```

```
Out[4]: 8
```

```
In [5]: compute_pref_at(669354,1635354)
```

```
Out[5]: 0
```

4.3.8 2.3.8 SVD dot

It is the dot product of svd of source and destination nodes

```
In [10]: def svd_dot(a,b):
             return np.dot(a,b)
```

4.4 2.4 Applying features

4.4.1 Reading a sample of Data from both train and test

```
In [7]: if os.path.isfile("data/after_eda/train_after_eda.csv"):
            df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv', nrows=100000, na
            df_final_train['indicator_link'] = pd.read_csv('data/train_y.csv', nrows=100000, na
            print("Our train matrix size ",df_final_train.shape)
```

```
Our train matrix size  (100000, 3)
```

```
In [8]: df_final_train.tail(2)
```

```
Out[8]:        source_node  destination_node  indicator_link
        99998       487275           1622261               1
        99999       552956            376409               1
```

```
In [4]: if os.path.isfile("data/after_eda/test_after_eda.csv"):
            df_final_train = pd.read_csv('data/after_eda/test_after_eda.csv', nrows=50000, name
            df_final_train['indicator_link'] = pd.read_csv('data/test_y.csv', nrows=50000, name
            print("Our train matrix size ",df_final_train.shape)
```

```
Our train matrix size  (50000, 3)
```

4.5 2.5 Adding a set of features

we will create these each of these features for both train and test data points

jaccard_followers
jaccard_followees
cosine_followers
cosine_followees
num_followers_s
num_followees_s
num_followers_d
num_followees_d
inter_followers
inter_followees

```
In [5]: if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
            #mapping jaccrd followers to train and test data
            df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                                    jaccard_for_followers(row['source_node'],r
            df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                                    jaccard_for_followers(row['source_node'],r

            #mapping jaccrd followees to train and test data
            df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                                    jaccard_for_followees(row['source_node'],r
            df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                                    jaccard_for_followees(row['source_node'],r


                #mapping jaccrd followers to train and test data
            df_final_train['cosine_followers'] = df_final_train.apply(lambda row:
                                                    cosine_for_followers(row['source_node'],ro
            df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                                    cosine_for_followers(row['source_node'],ro

            #mapping jaccrd followees to train and test data
            df_final_train['cosine_followees'] = df_final_train.apply(lambda row:
                                                    cosine_for_followees(row['source_node'],ro
            df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                                    cosine_for_followees(row['source_node'],ro

In [6]: def compute_features_stage1(df_final):
            #calculating no of followers followees for source and destination
            #calculating intersection of followers and followees for source and destination
            num_followers_s=[]
            num_followees_s=[]
            num_followers_d=[]
            num_followees_d=[]
            inter_followers=[]
            inter_followees=[]
```

```
        for i,row in df_final.iterrows():
            try:
                s1=set(train_graph.predecessors(row['source_node']))
                s2=set(train_graph.successors(row['source_node']))
            except:
                s1 = set()
                s2 = set()
            try:
                d1=set(train_graph.predecessors(row['destination_node']))
                d2=set(train_graph.successors(row['destination_node']))
            except:
                d1 = set()
                d2 = set()
            num_followers_s.append(len(s1))
            num_followees_s.append(len(s2))

            num_followers_d.append(len(d1))
            num_followees_d.append(len(d2))

            inter_followers.append(len(s1.intersection(d1)))
            inter_followees.append(len(s2.intersection(d2)))

        return num_followers_s, num_followers_d, num_followees_s, num_followees_d, inter_fo
```

```
In [7]: if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
            df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
            df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
            df_final_train['inter_followers'], df_final_train['inter_followees']= compute_featu

            df_final_test['num_followers_s'], df_final_test['num_followers_d'], \
            df_final_test['num_followees_s'], df_final_test['num_followees_d'], \
            df_final_test['inter_followers'], df_final_test['inter_followees']= compute_feature

            hdf = HDFStore('data/fea_sample/storage_sample_stage1.h5')
            hdf.put('train_df',df_final_train, format='table', data_columns=True)
            hdf.put('test_df',df_final_test, format='table', data_columns=True)
            hdf.close()
        else:
            df_final_train = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'train_df',mo
            df_final_test = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'test_df',mode
```

4.6 2.6 Adding new set of features

we will create these each of these features for both train and test data points

adar index
is following back
belongs to same weakly connect components
shortest path between source and destination

```
In [11]: if not os.path.isfile('data/fea_sample/storage_sample_stage2.h5'):
             #mapping adar index on train
             df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row[
             #mapping adar index on test
             df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['s

             #--------------------------------------------------------------------------------
             #mapping followback or not on train
             df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(ro

             #mapping followback or not on test
             df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row[

             #--------------------------------------------------------------------------------
             #mapping same component of wcc or not on train
             df_final_train['same_comp'] = df_final_train.apply(lambda row: belongs_to_same_wc

             ##mapping same component of wcc or not on train
             df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(

             #--------------------------------------------------------------------------------
             #mapping shortest path on train
             df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shorte
             #mapping shortest path on test
             df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_shortest

             hdf = HDFStore('data/fea_sample/storage_sample_stage2.h5')
             hdf.put('train_df',df_final_train, format='table', data_columns=True)
             hdf.put('test_df',df_final_test, format='table', data_columns=True)
             hdf.close()
         else:
             df_final_train = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'train_df',
             df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'test_df',mo
```

4.7 2.7 Adding new set of features

we will create these each of these features for both train and test data points

Weight Features
weight of incoming edges
weight of outgoing edges
weight of incoming edges + weight of outgoing edges
weight of incoming edges * weight of outgoing edges
2*weight of incoming edges + weight of outgoing edges
weight of incoming edges + 2*weight of outgoing edges
Page Ranking of source
Page Ranking of dest
katz of source
katz of dest

hubs of source
hubs of dest
authorities_s of source
authorities_s of dest

In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other. `credit` - Graph-based Features for Supervised Link Prediction William Cukierski, Benjamin Hamner, Bo Yang

$$W = \frac{1}{\sqrt{1 + |X|}} \tag{3}$$

it is directed graph so calculated Weighted in and Weighted out differently

```
In [17]: #weight for source and destination of each link
         Weight_in = {}
         Weight_out = {}
         for i in  tqdm(train_graph.nodes()):
             s1=set(train_graph.predecessors(i))
             w_in = 1.0/(np.sqrt(1+len(s1)))
             Weight_in[i]=w_in

             s2=set(train_graph.successors(i))
             w_out = 1.0/(np.sqrt(1+len(s2)))
             Weight_out[i]=w_out

         #for imputing with mean
         mean_weight_in = np.mean(list(Weight_in.values()))
         mean_weight_out = np.mean(list(Weight_out.values()))
```

```
100%|| 1780722/1780722 [00:10<00:00, 168740.44it/s]
```

```
In [10]: if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):
             #mapping to pandas train
             df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weig
             df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_

             #mapping to pandas test
             df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weigh
             df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_ou


             #some features engineerings on the in and out weights
             df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.weight_ou
             df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.weight_ou
             df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_final_train.weig
             df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_train.weig

             #some features engineerings on the in and out weights
             df_final_test['weight_f1'] = df_final_test.weight_in + df_final_test.weight_out
             df_final_test['weight_f2'] = df_final_test.weight_in * df_final_test.weight_out
             df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_final_test.weight_
             df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_

In [11]: if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):

             #page rank for source and destination in Train and Test
             #if anything not there in train graph then adding mean page rank
             df_final_train['page_rank_s'] = df_final_train.source_node.apply(lambda x:pr.get(
             df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr

             df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,
             df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x:pr.g
             #=================================================================================

             #Katz centrality score for source and destination in Train and test
             #if anything not there in train graph then adding mean katz score
             df_final_train['katz_s'] = df_final_train.source_node.apply(lambda x: katz.get(x,
             df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.g

             df_final_test['katz_s'] = df_final_test.source_node.apply(lambda x: katz.get(x,me
             df_final_test['katz_d'] = df_final_test.destination_node.apply(lambda x: katz.get
             #=================================================================================

             #Hits algorithm score for source and destination in Train and test
             #if anything not there in train graph then adding 0
             df_final_train['hubs_s'] = df_final_train.source_node.apply(lambda x: hits[0].get
             df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: hits[0

             df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].get(x
             df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hits[0].
             #=================================================================================

             #Hits algorithm score for source and destination in Train and Test
             #if anything not there in train graph then adding 0
             df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda x: hits
             df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x:

             df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1
             df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: h
             #=================================================================================

             hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
             hdf.put('train_df',df_final_train, format='table', data_columns=True)
             hdf.put('test_df',df_final_test, format='table', data_columns=True)
             hdf.close()
         else:
             df_final_train = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'train_df',m
             df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'test_df',mod
```

4.8 2.8 Adding new set of features

we will create these each of these features for both train and test data points

SVD features for both source and destination
Preferntial Attachment between two nodes

```
In [3]: def svd(x, S):
            try:
                z = sadj_dict[x]
                return S[z]
            except:
                return [0,0,0,0,0,0]
```

```
In [4]: #for svd features to get feature vector creating a dict node val and inedx in svd vect
        sadj_col = sorted(train_graph.nodes())
        sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

```
In [5]: if not os.path.isfile('data/fea_sample/storage_sample_stage4.h5'):
            #=====================================================================================

            df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'sv
            df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

            df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','sv
            df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
            #=====================================================================================

            df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'sv
            df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

            df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','sv
            df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
            #=====================================================================================

            df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd
            df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

            df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd
            df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

            #=====================================================================================

            df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd
```

```
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #=====================================================================================

    hdf = HDFStore('data/fea_sample/storage_sample_stage4.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
```

```
In [7]: #reading
        from pandas import read_hdf
        df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode=
        df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r
```

```
In [8]: df_final_train.columns
```

```
Out[8]: Index(['source_node', 'destination_node', 'indicator_link',
               'jaccard_followers', 'jaccard_followees', 'cosine_followers',
               'cosine_followees', 'num_followers_s', 'num_followees_s',
               'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
               'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
               'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
               'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
               'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
               'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
               'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
               'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
               'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
              dtype='object')
```

```
In [11]: #Calculating svd of train

         df_final_train["u_s"] = df_final_train[['svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u
         df_final_train["v_s"] = df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_
         df_final_train["u_d"] = df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u
         df_final_train["v_d"] = df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v
         df_final_train["svd_s"] = df_final_train.apply(lambda x: svd_dot(np.array(x["u_s"]), n
         df_final_train["svd_d"] = df_final_train.apply(lambda x: svd_dot(np.array(x["u_d"]), n
         df_final_train["svd_dot"] = df_final_train.apply(lambda x: svd_dot(x["svd_s"], x["svd_
         df_final_train.drop(columns=["u_s", "v_s", "u_d", "v_d", "svd_s", "svd_d"], inplace=Tr
```

```
In [12]: #Calculating svd of train

         df_final_test["u_s"] = df_final_test[['svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s
         df_final_test["v_s"] = df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_
         df_final_test["u_d"] = df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d
         df_final_test["v_d"] = df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d
```

```
        df_final_test["svd_s"] = df_final_test.apply(lambda x: svd_dot(np.array(x["u_s"]), np
        df_final_test["svd_d"] = df_final_test.apply(lambda x: svd_dot(np.array(x["u_d"]), np
        df_final_test["svd_dot"] = df_final_test.apply(lambda x: svd_dot(x["svd_s"], x["svd_d
        df_final_test.drop(columns=["u_s", "v_s", "u_d", "v_d", "svd_s", "svd_d"], inplace=Tru
```

```
In [13]: df_final_train.columns
```

```
Out[13]: Index(['source_node', 'destination_node', 'indicator_link',
                'jaccard_followers', 'jaccard_followees', 'cosine_followers',
                'cosine_followees', 'num_followers_s', 'num_followees_s',
                'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
                'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
                'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
                'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
                'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
                'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
                'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
                'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
                'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
                'svd_dot'],
               dtype='object')
```

```
In [14]: df_final_train["pref_attachment"] = df_final_train.apply(lambda row: compute_pref_at(r
         df_final_test["pref_attachment"] = df_final_test.apply(lambda row: compute_pref_at(row
```

```
In [15]: df_final_train.to_csv("Final_train.csv", index=False)
         df_final_test.to_csv("Final_test.csv", index=False)
```

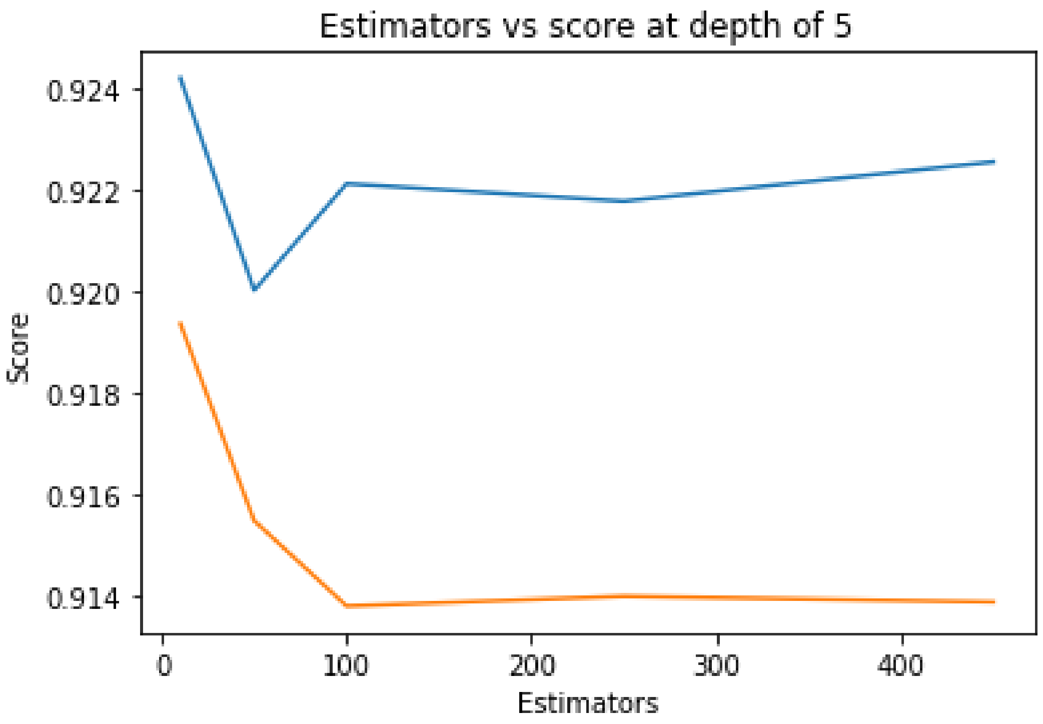
```
In [1]: #Importing Libraries
        # please do go through this python notebook:
        import warnings
        warnings.filterwarnings("ignore")

        import csv
        import pandas as pd#pandas to create small dataframes
        import datetime #Convert to unix time
        import time #Convert to unix time
        # if numpy is not installed already : pip3 install numpy
        import numpy as np#Do aritmetic operations on arrays
        # matplotlib: used to plot graphs
        import matplotlib
        import matplotlib.pylab as plt
        import seaborn as sns#Plots
        from matplotlib import rcParams#Size of plots
        from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
        import math
        import pickle
        import os
        # to install xgboost: pip3 install xgboost
```

```
import xgboost as xgb

import warnings
import networkx as nx
import pdb
import pickle
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

5 Models

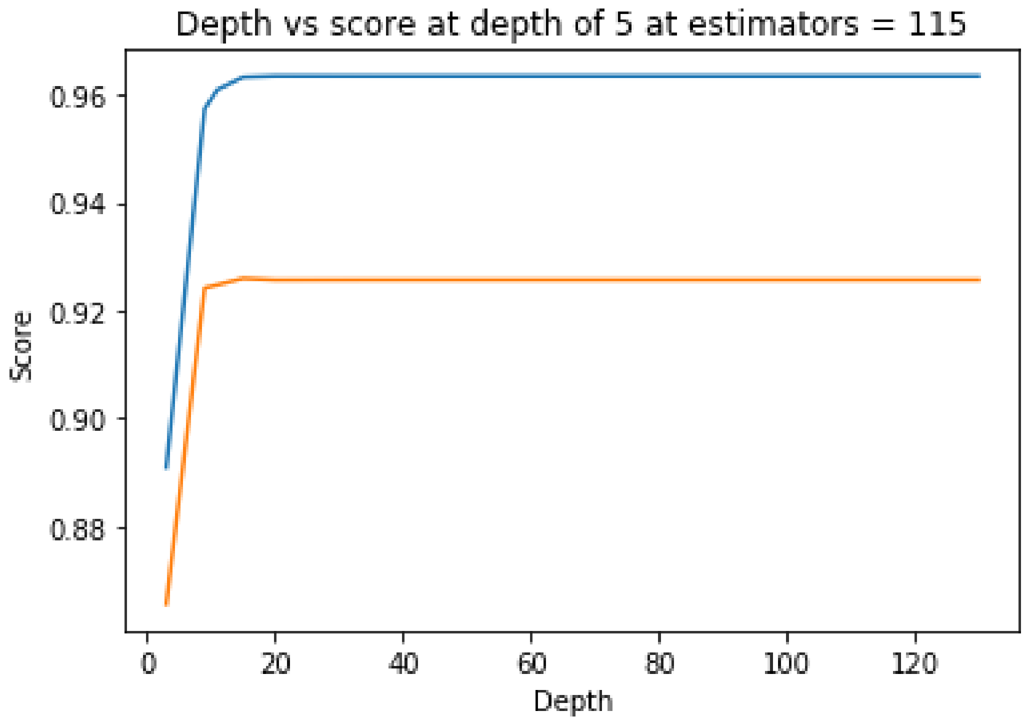
```
In [2]: df_final_train = pd.read_csv("Final_train.csv")
        df_final_test = pd.read_csv("Final_test.csv")

In [3]: y_train = df_final_train.indicator_link
        y_test = df_final_test.indicator_link

In [4]: df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplac
        df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace

In [5]: estimators = [10,50,100,250,450]
        train_scores = []
        test_scores = []
        for i in estimators:
            clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=5, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=52, min_samples_split=120,
                    min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,ve
            clf.fit(df_final_train,y_train)
            train_sc = f1_score(y_train,clf.predict(df_final_train))
            test_sc = f1_score(y_test,clf.predict(df_final_test))
            test_scores.append(test_sc)
            train_scores.append(train_sc)
            print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
        plt.plot(estimators,train_scores,label='Train Score')
        plt.plot(estimators,test_scores,label='Test Score')
        plt.xlabel('Estimators')
        plt.ylabel('Score')
        plt.title('Estimators vs score at depth of 5')

Estimators =  10 Train Score 0.9241865951294872 test Score 0.9193656653800973
Estimators =  50 Train Score 0.9200226410347792 test Score 0.9154938466393184
Estimators =  100 Train Score 0.9221138874084294 test Score 0.913818878571579
Estimators =  250 Train Score 0.9217698449191155 test Score 0.91401173427544
Estimators =  450 Train Score 0.9225414676463518 test Score 0.9138994756901307
```

```
Out[5]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



```
In [6]: depths = [3,9,11,15,20,35,50,70,130]
        train_scores = []
        test_scores = []
        for i in depths:
            clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=i, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=52, min_samples_split=120,
                    min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,
            clf.fit(df_final_train,y_train)
            train_sc = f1_score(y_train,clf.predict(df_final_train))
            test_sc = f1_score(y_test,clf.predict(df_final_test))
            test_scores.append(test_sc)
            train_scores.append(train_sc)
            print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
        plt.plot(depths,train_scores,label='Train Score')
        plt.plot(depths,test_scores,label='Test Score')
        plt.xlabel('Depth')
        plt.ylabel('Score')
        plt.title('Depth vs score at depth of 5 at estimators = 115')
        plt.show()
```

```
depth =  3 Train Score 0.890988741217303 test Score 0.8656639004149377
depth =  9 Train Score 0.9573384402249207 test Score 0.924173605854151
depth =  11 Train Score 0.960864574974861 test Score 0.9247628462654859
depth =  15 Train Score 0.9631702198838235 test Score 0.925944605556256
depth =  20 Train Score 0.9633714140644835 test Score 0.9256678525566807
depth =  35 Train Score 0.9633623753528707 test Score 0.9256803687567089
depth =  50 Train Score 0.9633623753528707 test Score 0.9256803687567089
depth =  70 Train Score 0.9633623753528707 test Score 0.9256803687567089
depth =  130 Train Score 0.9633623753528707 test Score 0.9256803687567089
```



```
In [7]: from sklearn.metrics import f1_score
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import f1_score
        from sklearn.model_selection import RandomizedSearchCV
        from scipy.stats import randint as sp_randint
        from scipy.stats import uniform

        param_dist = {"n_estimators":sp_randint(105,125),
                     "max_depth": sp_randint(10,15),
                     "min_samples_split": sp_randint(110,190),
                     "min_samples_leaf": sp_randint(25,65)}

        clf = RandomForestClassifier(random_state=25,n_jobs=-1)

        rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                           n_iter=5,cv=10,scoring='f1',random_state=25)

        rf_random.fit(df_final_train,y_train)
        print('mean test scores',rf_random.cv_results_['mean_test_score'])
        print('mean train scores',rf_random.cv_results_['mean_train_score'])

mean test scores [0.96184893 0.96189299 0.96041684 0.96182975 0.9629886 ]
mean train scores [0.96264252 0.96224303 0.96081531 0.96231698 0.96382023]
```

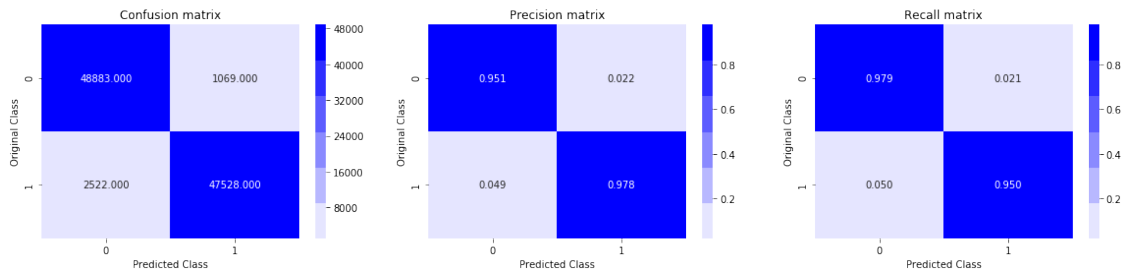
```
In [8]: clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=14, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=28, min_samples_split=111,
                    min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
                    oob_score=False, random_state=25, verbose=0, warm_start=False)

In [9]: clf.fit(df_final_train,y_train)
        y_train_pred = clf.predict(df_final_train)
        y_test_pred = clf.predict(df_final_test)

In [10]: from sklearn.metrics import f1_score
         print('Train f1 score',f1_score(y_train,y_train_pred))
         print('Test f1 score',f1_score(y_test,y_test_pred))

Train f1 score 0.9635974738207955
Test f1 score 0.9258535560458031
```

```
In [11]: from sklearn.metrics import confusion_matrix
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)

             A =(((C.T)/(C.sum(axis=1))).T)

             B =(C/C.sum(axis=0))
             plt.figure(figsize=(20,4))

             labels = [0,1]
             # representing A in heatmap format
             cmap=sns.light_palette("blue")
             plt.subplot(1, 3, 1)
```

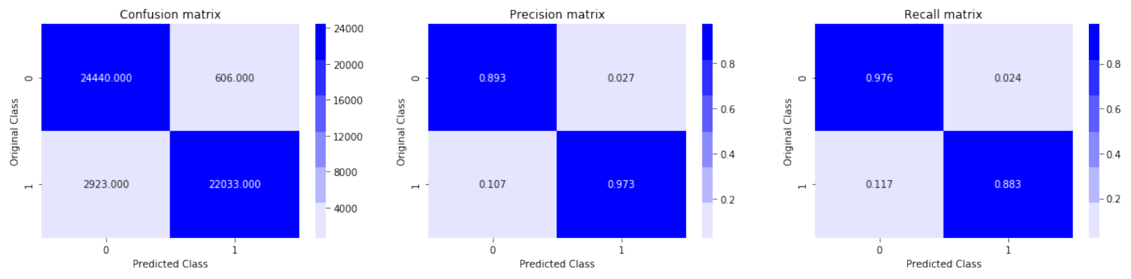
```
            sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=
            plt.xlabel('Predicted Class')
            plt.ylabel('Original Class')
            plt.title("Confusion matrix")

            plt.subplot(1, 3, 2)
            sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=
            plt.xlabel('Predicted Class')
            plt.ylabel('Original Class')
            plt.title("Precision matrix")

            plt.subplot(1, 3, 3)
            # representing B in heatmap format
            sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=
            plt.xlabel('Predicted Class')
            plt.ylabel('Original Class')
            plt.title("Recall matrix")

            plt.show()
```

```
In [12]: print('Train confusion_matrix')
         plot_confusion_matrix(y_train,y_train_pred)
         print('Test confusion_matrix')
         plot_confusion_matrix(y_test,y_test_pred)
```

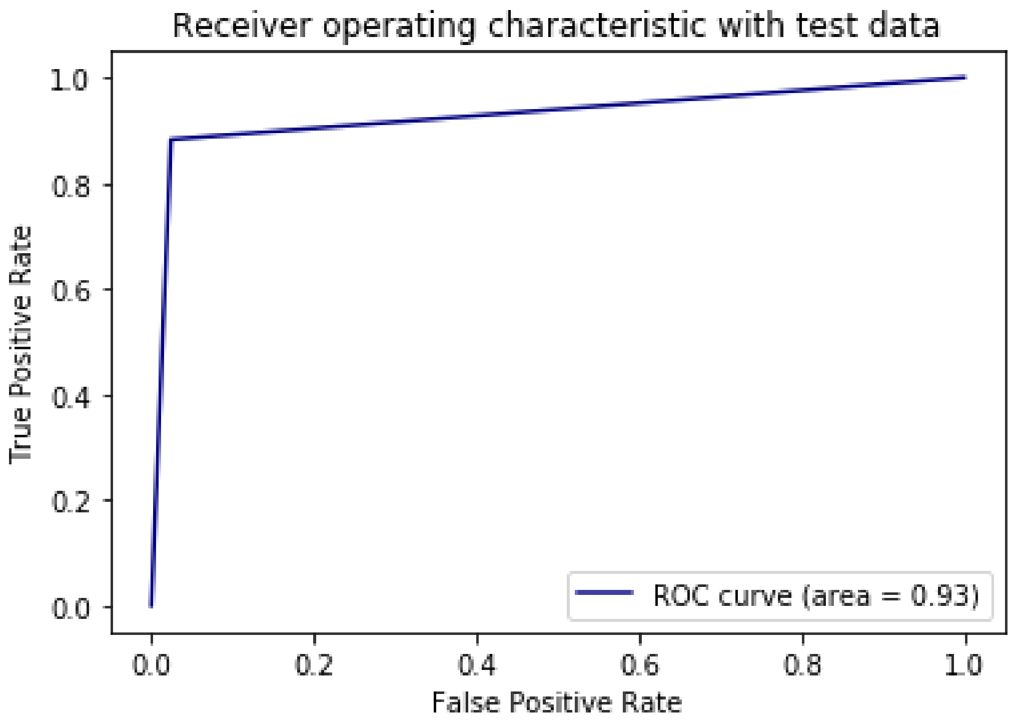
```
Train confusion_matrix
```



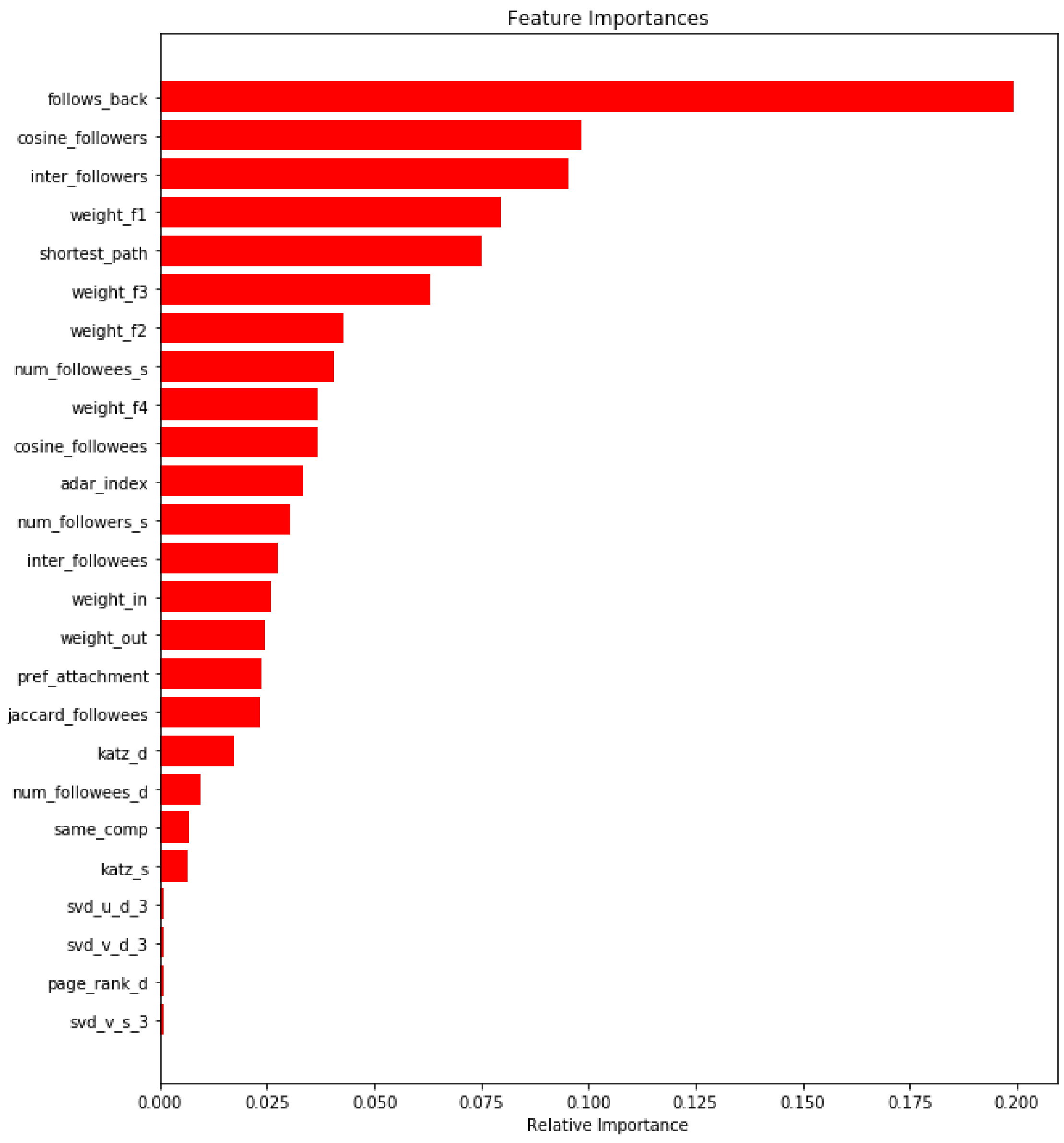
```
Test confusion_matrix
```



```
In [13]: from sklearn.metrics import roc_curve, auc
         fpr,tpr,ths = roc_curve(y_test,y_test_pred)
         auc_sc = auc(fpr, tpr)
         plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('Receiver operating characteristic with test data')
         plt.legend()
         plt.show()
```



```
In [14]: features = df_final_train.columns
         importances = clf.feature_importances_
         indices = (np.argsort(importances))[-25:]
         plt.figure(figsize=(10,12))
         plt.title('Feature Importances')
         plt.barh(range(len(indices)), importances[indices], color='r', align='center')
         plt.yticks(range(len(indices)), [features[i] for i in indices])
         plt.xlabel('Relative Importance')
         plt.show()
```



5.1 Using XGBoost

```
In [15]: from sklearn.model_selection import RandomizedSearchCV
         from xgboost import XGBClassifier

         params = {"learning_rate":[0.001, 0.01, 0.1, 1], "max_depth":[6, 9, 11],\
                   "gamma":[1, 2, 3, 4, 5], "min_child_weight":[1, 3, 4, 5, 7],\
                   "n_estimators":[50, 100, 150, 200, 250]}
         # params['objective'] = 'binary:logistic'
         # params['eval_metric'] = 'logloss'
         # params['eta'] = 0.02
```

```
xgb = XGBClassifier(objective = 'binary:logistic')
xgb_rand = RandomizedSearchCV(xgb, param_distributions = params, cv = 2,\
                              n_iter=5, scoring="roc_auc", verbose=3)
xgb_rand_fit = xgb_rand.fit(df_final_train, y_train)

#Optimal values from RandomizedSearchCV
opt_lr = xgb_rand_fit.best_estimator_.learning_rate
opt_md = xgb_rand_fit.best_estimator_.max_depth
opt_gamma = xgb_rand_fit.best_estimator_.gamma
opt_mcw = xgb_rand_fit.best_estimator_.min_child_weight
opt_nest = xgb_rand_fit.best_estimator_.n_estimators

print(f"\nOptimal learning rate = {opt_lr}")
print(f"Optimal max_depth = {opt_md}")
print(f"Optimal gamma = {opt_gamma}")
print(f"Optimal min_child_weight = {opt_mcw}")
print(f"Optimal n_estimators = {opt_nest}")
```

```
Fitting 2 folds for each of 5 candidates, totalling 10 fits
[CV] n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=1, gamma=5


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.


[CV]  n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=1, gamma=5, score=0.9979
[CV] n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=1, gamma=5


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   18.3s remaining:    0.0s


[CV]  n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=1, gamma=5, score=0.9980
[CV] n_estimators=100, min_child_weight=3, max_depth=6, learning_rate=1, gamma=1


[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   33.8s remaining:    0.0s


[CV]  n_estimators=100, min_child_weight=3, max_depth=6, learning_rate=1, gamma=1, score=0.998
[CV] n_estimators=100, min_child_weight=3, max_depth=6, learning_rate=1, gamma=1
[CV]  n_estimators=100, min_child_weight=3, max_depth=6, learning_rate=1, gamma=1, score=0.998
[CV] n_estimators=100, min_child_weight=7, max_depth=9, learning_rate=0.01, gamma=1
[CV]  n_estimators=100, min_child_weight=7, max_depth=9, learning_rate=0.01, gamma=1, score=0.
[CV] n_estimators=100, min_child_weight=7, max_depth=9, learning_rate=0.01, gamma=1
[CV]  n_estimators=100, min_child_weight=7, max_depth=9, learning_rate=0.01, gamma=1, score=0.
[CV] n_estimators=50, min_child_weight=3, max_depth=6, learning_rate=0.01, gamma=3
[CV]  n_estimators=50, min_child_weight=3, max_depth=6, learning_rate=0.01, gamma=3, score=0.9
```

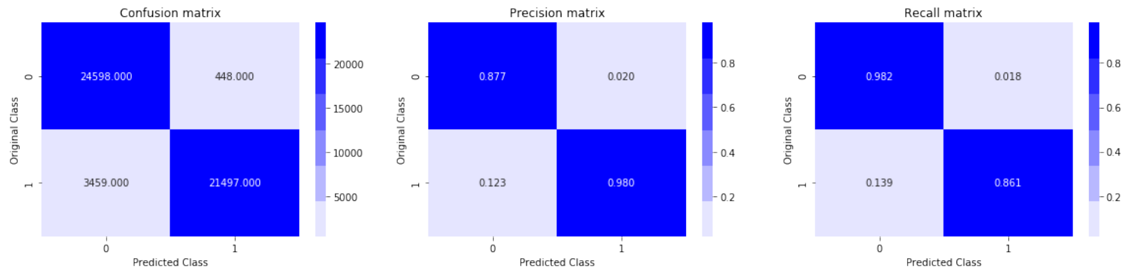
```
[CV] n_estimators=50, min_child_weight=3, max_depth=6, learning_rate=0.01, gamma=3
[CV]  n_estimators=50, min_child_weight=3, max_depth=6, learning_rate=0.01, gamma=3, score=0.98
[CV] n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=0.001, gamma=5
[CV]  n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=0.001, gamma=5, score=0.9
[CV] n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=0.001, gamma=5
[CV]  n_estimators=50, min_child_weight=1, max_depth=9, learning_rate=0.001, gamma=5, score=0.9


[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:  3.0min finished



Optimal learning rate = 1
Optimal max_depth = 6
Optimal gamma = 1
Optimal min_child_weight = 3
Optimal n_estimators = 100
```

```
In [16]: import xgboost as xgb
         params = {"objective": "binary:logistic", "eval_metric": "logloss", "learning_rate": o
                   "max_depth": opt_md, "gamma": opt_gamma, "min_child_weight": opt_mcw,\
                   "n_estimators":opt_nest}

         d_train = xgb.DMatrix(df_final_train, label=y_train)
         d_test = xgb.DMatrix(df_final_test, label=y_test)

         watchlist = [(d_train, 'train'), (d_test, 'valid')]

         bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, verbose_eva

         xgbdmat = xgb.DMatrix(df_final_train,y_train)
         predict_y = bst.predict(d_test)
```

```
[0]       train-logloss:0.240611       valid-logloss:0.267669
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]       train-logloss:0.049471       valid-logloss:0.405448
[20]       train-logloss:0.033593       valid-logloss:0.533592
Stopping. Best iteration:
[1]       train-logloss:0.142228       valid-logloss:0.221139
```

```
In [17]: predicted_y =np.array(predict_y>0.5,dtype=int)
         print("Total number of data points :", len(predicted_y))
         plot_confusion_matrix(y_test, predicted_y)
```

```
Total number of data points : 50002
```



5.2 Procedures taken:

- The given problem statement was to find missing links to recommend users
- Various graph features are added
- The features are then visualized with various plots
- The dataset is then split into train and test
- Different ML models (RandomForest and Xgboost) are carried out and hyperparameter tuning is performed on XGBoost.